

Manuscript version: Author's Accepted Manuscript

The version presented in WRAP is the author's accepted manuscript and may differ from the published version or Version of Record.

Persistent WRAP URL:

<http://wrap.warwick.ac.uk/108585>

How to cite:

Please refer to published version for the most recent bibliographic citation information. If a published version is known of, the repository item page linked to above, will contain details on accessing it.

Copyright and reuse:

The Warwick Research Archive Portal (WRAP) makes this work by researchers of the University of Warwick available open access under the following conditions.

Copyright © and all moral rights to the version of the paper presented here belong to the individual author(s) and/or other copyright owners. To the extent reasonable and practicable the material made available in WRAP has been checked for eligibility before being made available.

Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge. Provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.

Publisher's statement:

Please refer to the repository item page, publisher's statement section, for further information.

For more information, please contact the WRAP Team at: wrap@warwick.ac.uk.

An asynchronous method for cloud-based rendering

Keith Bugeja · Kurt Debattista · Sandro Spina

Received: date / Accepted: date

Abstract Interactive high-fidelity rendering is still unachievable on many consumer devices. Cloud gaming services have shown promise in delivering interactive graphics beyond the individual capabilities of user devices. However, a number of shortcomings are manifest in these systems: high network bandwidths are required for higher resolutions and input lag due to network fluctuations heavily disrupts user experience. In this paper we present a scalable solution for interactive high-fidelity graphics based on a distributed rendering pipeline where direct lighting is computed on the client device and indirect lighting in the cloud. The client device keeps a local cache for indirect lighting which is asynchronously updated using an object space representation; this allows us to achieve interactive rates that are unconstrained by network performance for a wide range of display resolutions that are also robust to input lag. Furthermore, in multi-user environments, the computation of indirect lighting is amortised over participating clients.

Keywords Rendering · Rasterization · Global Illumination · Distributed Algorithms · Cloud Computing

1 Introduction

An accurate lighting model and rapid visual feedback are two important factors in conveying realistic graphics at interactive rates [33][41]. While a large number of methods for computing realtime dynamic global illumination have been proposed, increasing the level of immersion in virtual environments, these typically trade

off quality and accuracy for performance, and are incapable of running on all but the most powerful machines.

Cloud computing, which has enabled the use of low performance devices for tasks beyond their computational capabilities, has been gainfully employed to accelerate rendering in both offline and realtime settings. In offline rendering, services such as Render Rocket [2] or Autodesk 360 [1] make available an on-demand infrastructure for distributed rendering in the cloud, essentially providing each individual user with a Render Farm for accelerating image synthesis. In the latter case, cloud computing has heralded the rise of interactive streaming services for video games (cloud gaming systems), which allow a wide range of devices to consume games that are either unavailable on a given platform or too taxing for its hardware to run satisfactorily, making the computational capacity of client devices largely irrelevant [28]. Although effective in providing a homogeneous experience to a plethora of devices with varying capabilities, the cloud gaming paradigm is highly susceptible to network latency and bandwidth constraints. High definition (HD) and ultra high definition (UHD) streams, especially at higher frame rates, transfer significant amounts of data which may exclude users with low-tier network configurations from using the service due to bandwidth limitations (see Table 1). Furthermore, disruptions caused by fluctuations in network performance negatively affect the user experience. A comparison of cloud gaming systems carried out by Digital Foundry [5] has shown that load and network performance fluctuations can significantly increase input lag, making a game less responsive; in some tests, the recorded lag was as high as 300ms, higher than the threshold of 250ms at which games are virtually unplayable.

K. Bugeja
Department of Computer Science,
University of Malta
E-mail: keith.bugeja@um.edu.mt

Table 1: Bandwidth requirements/recommendations for various video-on-demand and cloud gaming services.

Service	Resolution	Bandwidth
Netflix	576p	2.0 Mbit/s
Netflix	720p	4.0 Mbit/s
Netflix	1080p	5.0 Mbit/s
Netflix	2160p	25.0 Mbit/s
Hulu Plus	576p	1.0 Mbit/s
Hulu Plus	720p	2.0 Mbit/s
Hulu Plus	1080p	3.2 Mbit/s
OnLive	576p	2.0 Mbit/s
OnLive	720p	5.0 Mbit/s
Playstation Now	720p	5.0 Mbit/s
Gamefly Streaming	720p30	5.0 and 10.0 Mbit/s
GeForce NOW	720p60	10.0 and 20.0 Mbit/s
GeForce NOW	1080p60	10.0 and 50.0 Mbit/s

Typical cloud gaming systems, commercial ones also, employ a straightforward offloading strategy where each connecting client is synchronously mapped to a single cloud server running the desired application. Virtualised environments are oftentimes employed for less computationally-demanding applications such as legacy titles. Application output is then streamed back to the client as video [10]. Besides lacking the scalability required of economic cloud deployments, this approach does not take advantage of amortisation of computation since each server works in isolation.

This paper proposes Remote Asynchronous Indirect Lighting (RAIL), a distributed rendering technique that decouples inexpensive computations from the rest of the rendering pipeline and moves their execution to the client device. Through asynchronous computation, the system provides highly responsive high-fidelity rendering at HD resolutions and higher, and makes use of amortisation of computation to deliver improved scalability for multiple users sharing the same virtual environments [8]. The contributions of this work are:

- a scalable asynchronous distributed rendering method with low bandwidth requirements that is resolution-independent and robust to network service fluctuations, expected of typical connectivity over the Internet;
- the application of the method to the software-as-a-service (SaaS) paradigm for highly interactive rendering in HD (and higher) over a wide spectrum of devices.

2 Related Work

Dachsbacher et al. [12] present a survey of many-lights algorithms, a class of global illumination methods inspired by Instant Radiosity [22], a bidirectional

rendering technique that emits rays from light sources and follows them as they interact with surfaces across the scene. At every ray-surface interaction, a virtual point light (VPL) is created, to approximate the indirect lighting contribution of the surface region centred at the point of intersection. Both direct and indirect lighting computation are reduced to a single operation, the summation of contributions from point light sources dotted across the scene. Debattista et al. [15] introduced a hybrid of Instant Radiosity [22] and the Irradiance Cache [42], which is conceptually similar to the shading method proposed in this work, albeit the former is view-dependent. Bikker et al. [6] present a method to generate a view-independent point cloud for storing shading information; we use a similar method to generate a representation for static scene geometry, which is adequate to demonstrate the concept of remote asynchronous rendering. Nevertheless, RAIL is not tied to a specific generation algorithm and more efficient ones may be used if so desired.

An overview of parallel rendering algorithms for high-fidelity graphics can be found in Chalmers et al. [9]. In the context of interactive distributed rendering in the cloud, Pajak et al. [34] proposed a remote rendering service based on a single desktop setup that couples rendering, compression and streaming. In order to aid image reconstruction at the client from low resolution frames and increase robustness to data loss, they augment the streaming of video information using depth and motion information, achieving frame rates of 25 to 30 Hz at SVGA resolutions (800×600) and 9 to 12 Hz at HD resolutions (1920×1080) on a weak client (notebook with a low-end GPU). Crassin et al. [10] propose Cloudlight, a distributed rendering pipeline for global illumination, focusing on low network latency and the amortisation of computation over multi-user virtual environments. As opposed to rendering entirely in the cloud, they use an approach similar to what we propose, where the rendering pipeline is only partially offloaded from the client. In particular, they introduce a distributed rendering pipeline which computes the indirect lighting contribution in the cloud, amortising the computations across multiple clients in multi-user settings. Three lighting algorithms were proposed, each with different bandwidth and reconstruction costs [11][31][19][29]. Two of these algorithms, the path-traced irradiance maps and realtime photon mapping, are asynchronous in nature, decoupling client updates from the cloud computation and the network performance. Irradiance maps yield low bandwidth requirements, and reconstruction costs are also cheap, but the difficulty in acquiring UV-parameterisation for moderately complex scenes doesn't always make

them a viable option due to the laborious nature of the parameterisation [10]. Photon tracing doesn't require any parameterisations but has substantially larger bandwidth requirements, close to an order of magnitude more than the requirements of streaming cloud gaming platforms. Moreover, the indirect lighting reconstruction at the client poses prohibitive computational costs for some low to mid-range devices. The third algorithm, which adopts a synchronous approach, uses cone-traced sparse voxel global illumination, and although updates at 30 Hz can be sustained for 5 clients, this soon drops to 12 Hz as soon as the number of clients is increased to 24. Liu et al. [27] propose a distributed rendering pipeline which, similarly to Cloudlight [10], streams GI information to the client using H.264 encoded video. Indirect lighting is computed using view-dependent techniques, such as reflective shadow maps [13], light propagation volumes [21] and voxel cone tracing [11]; thus, computation cannot be amortised across multiple clients and camera changes require GI to be recomputed.

3 Method

The goal of RAIL is to provide high-fidelity graphics that go beyond the capabilities of individual client devices, and to do so at highly interactive rates with minimal bandwidth requirements and input lag, which has been shown to be detrimental to user experience.

In high-fidelity rendering, dynamic indirect lighting comprises a substantial part of the computational cost of frame synthesis, beyond the means of most consumer devices. Conversely, direct lighting, can be adequately computed by most consumer devices furnished with basic hardware graphics capabilities, from smartphones to tablets, to laptops to desktop machines. These systems are equally capable of visualising precomputed indirect illumination, typically reconstructing it from view-independent data structures.

RAIL decouples direct and indirect lighting, computing the latter in the cloud. Indirect lighting is stored in a view-independent data structure, a snapshot of which is also kept by the client and asynchronously updated. This snapshot is used to reconstruct indirect lighting at every frame, whenever direct lighting is computed by the client. This method has a number of advantages: (i) visual fidelity may go beyond the computational possibilities of a client device in isolation; (ii) frame updates are not bound to network performance, eliminating any input lag; and (iii) a server-based view-independent indirect lighting representation can be shared among multiple clients

collaborating within a virtual environment, possibly amortising computation costs.

Algorithm 1 Synchronous version of the proposed rendering algorithm.

```

1:  $Q \leftarrow \text{GeneratePointSet}(\text{scene})$  ▷ (§3.1)
2: while true do
3:    $V \leftarrow \text{TraceVPLs}(\text{scene})$  ▷ (§3.2)
4:    $\text{ShadePointSet}(Q, V)$  ▷ (§3.2)
5:    $I \leftarrow \text{ReconstructIndirect}(\text{scene}, Q)$  ▷ (§3.2)
6:    $D \leftarrow \text{ComputeDirect}(\text{scene})$ 
7:    $\text{MergeAndPresent}(I, D)$  ▷ (§3.3)
8: end while

```

Although RAIL is designed as an asynchronous distributed rendering algorithm (see Figure 1), for clarity we initially present a synchronous centralised formulation of the algorithm, elaborating on how a point description of the scene is generated and used to compute indirect lighting for object surfaces and reconstruct the full global illumination solution in realtime (see Algorithm 1). Briefly, a point cloud representation of the scene is generated offline for sampling diffuse indirect lighting (line 1). Light tracing is used to generate VPLs (line 3), which are used to shade the generated point cloud (line 4). Indirect lighting is reconstructed from the sparse samples in the point cloud (line 5) and merged with the direct lighting component (lines 6-7). In the reconstruction of indirect lighting, static geometry (non-movable, non-deformable geometry) is shaded using interpolated irradiance from the point cloud (§3.3), while dynamic geometry is shaded using a higher-order ambient function, a coarse approximation of indirect lighting (§3.4). The asynchronous distributed formulation of the algorithm is introduced subsequently, in Section 4.

3.1 Generation of Diffuse Indirect Sample Point Set

A point representation of the scene is generated to sparsely sample the indirect lighting function for reconstruction during interactive rendering. This process is a one-time, per-scene precomputation step. The selection of indirect diffuse sample points in this work is based on Bikker et al. [6], who use a method similar to Brouillat et al. [7] where light tracing is employed to place a number of dart sources in the scene. A dart throwing technique is then applied, using these sources, to trace and record points on the surface of scene geometry. The dart density is controlled via poisson disk sampling [17], to ensure that an area is neither too densely nor too sparsely populated; the generated distribution is not necessarily maximal [18].

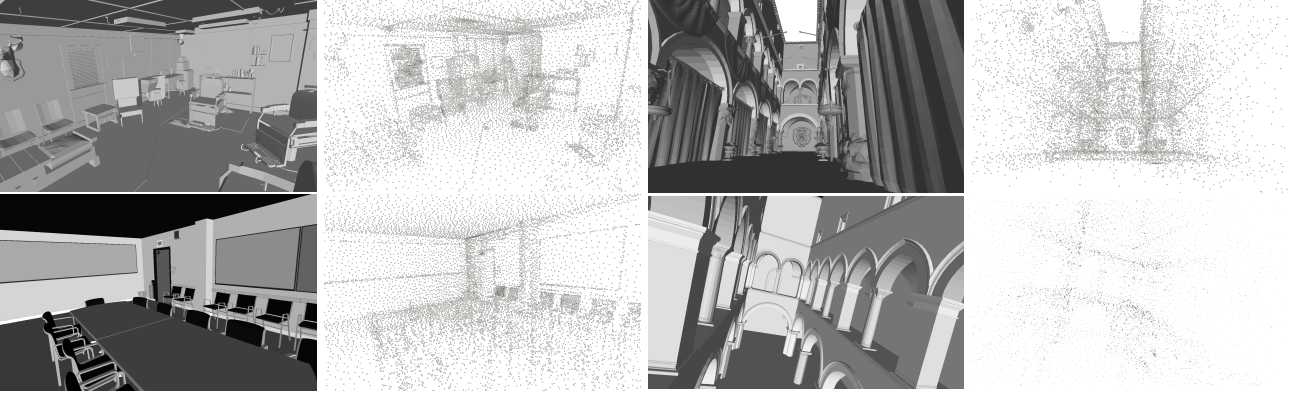


Fig. 2: Parallel view of geometry (left) and generated point sets (right) for the scenes used in this work.

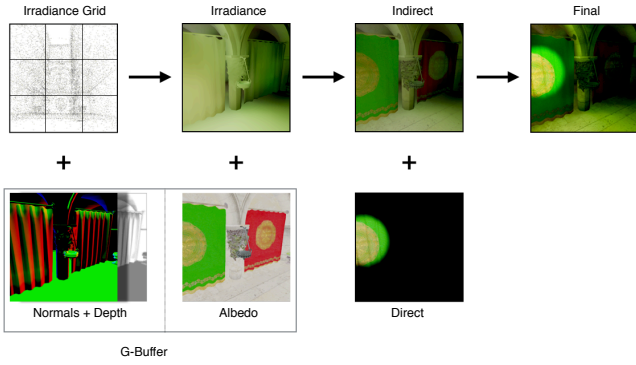


Fig. 3: The client rendering pipeline reconstructs indirect lighting from the shading point cloud and G-buffer, merging the result with direct lighting to obtain a GI solution.

3.3 Integrating Direct and Indirect Lighting

Reconstructing the indirect lighting contribution requires knowledge of point cloud Q and geometric details about the points for which the function is being reconstructed, such as surface positions, normals and albedos. In deferred shading pipelines, which perform screen-space shading, this information is readily available as a geometry-buffer (G-buffer) [16][38]. This makes reconstruction using Equation 2 straightforward but inefficient, since all points in Q have to be considered, even though they might not contribute anything to the final value. Thus, a multiple-reference regular grid G is introduced as an overlay on Q to accelerate nearest neighbour queries using spatial hashing. Records are referenced from each cell that overlaps their sphere of validity, simplifying the lookup to the examination of a single cell and the records contained within.

From equations 1 and 2, it follows that in order to calculate the contribution of each sample \mathbf{q} , properties

like position, surface normal, irradiance and range are required and thus, have to be accounted for in the space complexity of the regular grid. Specifically, let c be the number of cells along an edge of the grid and c^3 the total number of cells in the grid. The edge of each cell is c_l units long. The sample range of effect r in the weighting function (Equation 1) determines the spatial extent of a search operation and can be used to estimate the maximum number of references for a sample to $(2r / c_l)^3$, with the total being $|Q| \cdot (2r / c_l)^3$. Each cell holds an index to a record reference, which points to the first record in a bin that holds references to records affecting the given cell. A further indirection exists, that maps each reference to the actual record index, and finally the records themselves are stored. The index held in each cell of the grid is 4 bytes long, while each entry in the reference map is 8 bytes long. A single record is 23 bytes long, 12 bytes for position, 8 for surface normals and 3 for irradiance. The upper bound on the space requirements of the regular grid is thus $|Q| \cdot (8(2r / c_l)^3 + 23) + 4c^3$. A typical grid ($c = 64, c_l = 4, r = 6$ and $|Q| = 40000$) requires 10 MB of GPU memory.

The records are stored in memory as a structure-of-arrays (SoA) rather than an array-of-structures (AoS), the motivation being that only irradiance values in records change and thus locality during host-to-adaptor copies is exploited by modifying in bulk only the affected array. Thus, using a G-buffer and a regular grid storing irradiance samples, the diffuse indirect lighting for the current view can be efficiently reconstructed. High-frequency texture details are not present in the reconstruction and must be added via multiplicative blending of the albedo G-buffer channel and irradiance. Subsequently, the result is blended with the direct lighting output of the rendering pipeline. For devices with limited fill rates [26], the reconstructed irradiance channel may be smaller than the size of the frame buffer, to preserve high frame rate interactivity. In

these cases, prior to the final composition, geometry-aware upscaling using joint bilateral upsampling [23] is applied to the channel. Similarly to McGuire et al. [30], the weights used are based on 2D bilinear interpolation, normals and depth differences between the low and high resolution G-buffers. The upscaled irradiance is combined with the albedo channel and then with the direct contribution, to derive the final image (Figure 3).

Algorithm 2 Propagation of ambient contribution values to empty cells.

```

1: procedure PROPAGATE(grid, iterations)
2:    $A_c[\cdot \cdot \cdot] \leftarrow 1$ 
3:   for  $1 \leq n \leq \text{iterations}$  do
4:     for each  $\mathbf{p} \in \text{grid} \wedge \neg \text{isEmpty}(\mathbf{p})$  do
5:        $\text{cells} \leftarrow \text{FilterNeighbours}(\mathbf{p})$ 
6:       for each  $\mathbf{c} \in \text{cells} \wedge \text{isEmpty}(\mathbf{c})$  do
7:          $A_e[\mathbf{c}] \leftarrow A_e[\mathbf{c}] + A_e[\mathbf{p}]$ 
8:          $\text{inc } A_c[\mathbf{c}]$ 
9:       end for
10:    end for
11:    for each  $\mathbf{p} \in \text{grid}$  do
12:       $A_e[\mathbf{p}] \leftarrow A_e[\mathbf{p}] / A_c[\mathbf{p}]$ 
13:    end for
14:  end for
15: end procedure
16: procedure FILTERNEIGHBOURS( $\mathbf{p}$ )
17:    $\mathbf{n} \leftarrow A_n[\mathbf{p}]$ 
18:    $\text{cells} \leftarrow \emptyset$ 
19:   for each  $\text{axis} \in \{x, y, z\}$  do
20:     if  $|n_{\text{axis}}| \geq \frac{1}{\sqrt{3}}$  then
21:        $n_{\text{sgn}} = \text{sgn}(n_{\text{axis}})$ 
22:        $\text{cells} \leftarrow \text{cells} \cup \text{neighbour}(\mathbf{p}, \text{axis}, n_{\text{sgn}})$ 
23:     end if
24:   end for
25:   return  $\text{cells}$ 
26: end procedure

```

3.4 Dynamic Scenes

The system currently supports dynamic objects, deformable ones also. These objects are factored in the VPL tracing and point cloud shading steps (§3.2), and thus, can occlude and reflect light. To avoid changing the structure of the grid G (see §3.3) whenever an object moves, a coarser approximation of indirect lighting is used, inspired by ambient occlusion, and is computed in two steps. The constant ambient term, traditionally used to approximate indirect reflections in the scene, is turned into a higher-order function of space; this is then evaluated by partitioning the scene into a coarse regular grid, with each cell containing an ambient term approximation for the region, and trilinearly interpolating these values across adjacent cells

(see §3.5). The ambient term for each cell is computed from the weighted mean of all irradiance points in Q affecting the cell. This coarse grid is referred to as the *ambient grid* A , and is distinct from the grid G discussed in Section 3.3; the latter is used to accelerate nearest neighbour searches of Q . Spatially, A is fitted over scene geometry such that the longest edge of the bounding volume of the scene corresponds to an edge of the bounding volume of A . The edges of the ambient grid are equal, which means that, geometrically, it is a cube, subdivided equally along all edges. The constituting cells contain three important values, a single quantity A_e representing the weighted mean irradiance (*ambient contribution*) of the sample points in Q which are also contained within the volume of the cell itself, a normalised vector A_n representing the principal direction of the sample normals contained within the cell, and a scalar contribution count A_c that keeps track of the number of ambient contributions a given cell has received from neighbouring cells. Cells are indexed via a triple $\mathbf{p} = (x, y, z)$, where $0 \leq x, y, z < \text{subdivisions}$. The ambient contribution for a cell is computed as follows:

$$A_e[x, y, z] = \frac{\sum_{\mathbf{q} \in Q_{xyz}} \mathbf{q}_o \mathbf{q}_e}{\sum_{\mathbf{q} \in Q_{xyz}} \mathbf{q}_o}, \quad (3)$$

where Q_{xyz} is the subset of points in Q that is contained in the grid cell with index (x, y, z) , and \mathbf{q}_e and \mathbf{q}_o are the irradiance and ambient occlusion values for sample \mathbf{q} respectively. Note that \mathbf{q}_o is computed offline, during the generation of Q , and stored (see §3.1). The layout of scene geometry, and consequently the distribution of Q , may be such that some cells in the grid are empty (i.e., $Q_{xyz} = \emptyset$). A straightforward iterative approach is used to propagate the ambient contribution from neighbouring cells and fill empty ones. When the ambient grid is first created, principal component analysis is performed on the normal vectors of the samples contained in each cell. A principal component is determined and used as the aggregated surface normal A_n for that cell. Propagation is a runtime process which uses the ambient contribution values of neighbouring cells to populate the empty ones; the process is described in Algorithm 2. For each cell in the grid that has a valid ambient contribution (*source cell*), the direction of propagation is determined from the principal normal A_n . The normal is used to determine which neighbouring cells to consider. The principal normal $\mathbf{n} = A_p[p_x, p_y, p_z]$ is decomposed into its axial components n_x , n_y and n_z . If the length of each component exceeds a given threshold ($\pm \frac{1}{\sqrt{3}}$), then the cells adjacent to the face with the component axis' normal are added to the propagation set.

Consequently, the ambient contribution of the source cell is combined with that at each of the selected neighbours; A_c keeps track of the number of contributions a cell has received during one iteration of propagation. An iteration completes once all the source cells have been considered, after which the propagated contributions are averaged. Figure 4 gives an example of propagation, for two iterations. The threshold value $\frac{1}{\sqrt{3}}$ is the length of each component of a unit vector which makes an angle of $\pi/4$ to each principal axis.

3.5 Using the Ambient Grid

During rendering, the ambient grid A is used to contribute indirect lighting to regions that are not covered by the samples in Q , such as dynamic geometry. Irradiance at \mathbf{p} is coarsely estimated from the ambient contributions by performing trilinear interpolation between adjacent cells of the grid, denoted by $F_a(A, \mathbf{p})$. The application of F_a to geometry subject only to direct lighting can be seen in figures 5a and 5b. In the first figure, the object is illuminated only by means of direct light; since the light is partially occluded, a great part of the object is depicted in black. In the second figure the black patches on the object are now lit via the ambient function, albeit the latter does not take into account surface occlusion. Thus, a heavily occluded point receives the same contribution as one that is less occluded, leading to a loss of perception of the shape of the object, as can be observed in Figure 5b. In order to curtail a point's exposure to ambient lighting and provide a better perception of the shape of geometry [24], ambient occlusion (AO) is applied to F_a . The extension to the ambient function is thus:

$$F_{ao}(A, \mathbf{p}) = \frac{F_a(A, \mathbf{p})}{\pi} \int_{\Omega} V(\mathbf{p}, \omega) \omega \cdot N_p \, d\omega. \quad (4)$$

AO being a global method, it requires access to scene geometry in order to compute point visibility, which makes it less suitable for use in rasterisation. Instead of traditional AO, screen space ambient occlusion (SSAO) is used, which is a faster approximation that computes occlusion from neighbouring pixel depths rather than scene geometry [32]. The application of the new ambient function F_{ao} can be seen in Figure 5c, where the AO term is evaluated using SSAO.

4 Distributing the Rendering Pipeline

In this section, the synchronous rendering method described thus far is transformed into an asynchronous distributed rendering pipeline (see Algorithm 1).

The first three steps (lines 1-4) are computationally expensive, beyond the reach of low-end hardware such as smartphones and tablet devices, making them good candidates for offloading to a powerful server backend. Particularly, the first step (line 1), which generates the point set, is a one-time precomputation step that can be carried out offline. The last two steps (lines 5-7) may be easily run on a low-end device, provided Q is available, in the form of the regular grid G (see §3.3). Distributing the rendering pipeline, thus, becomes a problem of synchronisation, whereby the regular grids at client ends are made consistent with that at the server, and the server's representation of dynamic objects in the scene, such as light sources, is made consistent with that of its clients. The highly interactive nature of client applications precludes the use of a blocking synchronisation mechanism that depends on network performance. Thus, a particular grid G_c at a client device is treated as a local cache of the server version G_s , and is updated asynchronously, without affecting the local rendering steps (lines 5-7), which are allowed to run unconstrained. Server-side, G_s is updated to reflect indirect diffuse lighting in the scene (3-4) and executes independently of client communication. Any scene changes received from clients are queued and applied to scene state on the server, invalidating indirect lighting computed thus far (see Figure 1).

4.1 Synchronisation of Indirect Lighting

The first message exchange between a client and the server backend is the initial transfer of grid G_s to the client, such that $G_c = G_s$. Subsequent exchanges are always started by the client device, which sends camera details and any changes to scene state that potentially affect indirect lighting, such as changes in light sources. In response, the server uses the client's camera parameters to form a frustum and clip the point cloud. The points contained in the frustum are then sent back to the client, together with any changes performed to scene state by other clients. Frustum clipping operates at cell-based granularity; all points contained in a cell intersecting the frustum are included in the response message, provided the irradiance values of at least one of them has changed. Particularly, the structure of a response contains a unique identifier for the cell, followed by the irradiance triples for the points contained in that cell. Since the point ordering is deterministic, indices are not included with the irradiance values; a map at the client associates the position of each irradiance value for a received cell to a position in the SoA (see §3.3). This significantly reduces the size of data

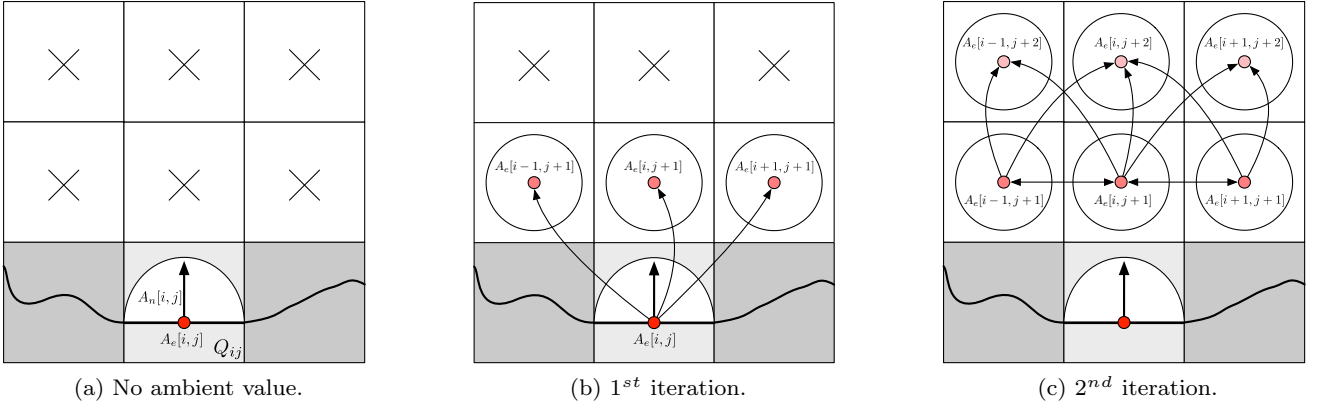


Fig. 4: Iterative propagation process for ambient lighting. In 4a, cells that contain no samples (and hence no ambient value) are marked in white. $A_e[i, j]$ marks the weighted mean of irradiance values, which is used to propagate indirect lighting to adjacent cells. In 4b, one iteration of propagation has been performed, and the empty cells adjacent to (i, j) which face the hemisphere of directions around $A_n[i, j]$ have received indirect lighting. Particularly, $A_n[i, j]$ acts as an occluder of sorts, to stop propagating values in its opposite direction. Cells that do not have a normal vector defined due to being empty of irradiance samples propagate values in all directions, that is, to all empty adjacent cells. 4c shows the ambient grid after two iterations of propagation, where all the empty cells are now populated with an ambient value.

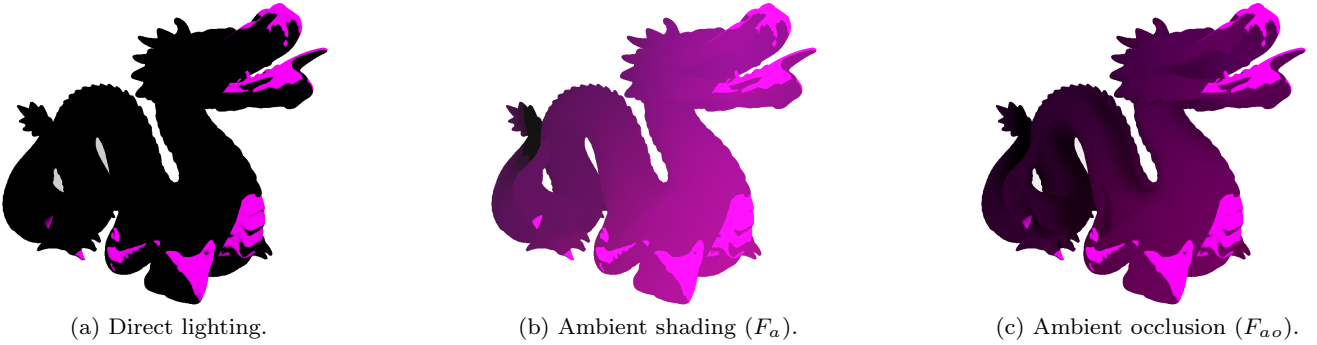


Fig. 5: Indirect lighting function for dynamic objects, where 5a is the base direct lighting, 5b shows the object shaded with the spatially-varying ambient term, and 5c augments 5b with screen space ambient occlusion.

transfers between the server and a client. Messages are packed using an LZF compressor [25], for a reduction in size of up to 40%.

4.2 Two-stage smoothing

The indirect lighting synchronisation process is asynchronous and can run at a lower frequency than the client rendering. In some scenes it can be very difficult to generate paths between light sources and the camera [4], requiring a larger number of VPLs to be traced for a more accurate estimation of indirect lighting [14]. This situation is exacerbated when the light sources are moving and any accumulated contribution has to

be constantly reset, leading to artefacts and flickering. To reduce flickering and the disparity between direct and indirect lighting update rates, a simple two-stage smoothing mechanism is used. The first stage uses exponential averaging to combine the current irradiance values in G_c with the newly received values from G_s such that for every sample, $E'_c = w_c E_c + (1 - w_c) E_s$, where E'_c is the updated irradiance value, E_c is the previous value held in the client cache G_c , and E_s is the updated value for the same sample. The weight w_c determines how quickly the local cache transitions to the server version. Increasing w_c exploits temporal coherence and reduces artefacts introduced by sudden illumination changes, making for a smooth transition. However, as a side effect of the slower transition, indirect

illumination may be perceived to be lagging behind the direct illumination. The weight w_c can be adjusted at runtime and is typically initialised to 0.5. An attempt has been made to base w_c on the change in orientation of the observer, with large changes resulting in a smaller weight and vice versa, and although preliminary results look promising this has not been formally evaluated. The second smoothing stage is used to compensate for the difference in update frequencies between direct and indirect lighting. For an arbitrary sample, the irradiance value used in the previous frame to reconstruct indirect lighting (E_d) and the most recent update for that same sample (E_c) are linearly interpolated to give the impression that irradiance is updating at the same rate as direct lighting. Let $E_r(t)$ be an interpolation function:

$$E_r(t) = \begin{cases} E_d & t \leq 0 \\ E_r(0) + \frac{t}{\Delta T}(E_r(\Delta T) - E_r(0)) & 0 < t < \Delta T \\ E_c & t \geq \Delta T \end{cases} \quad (5)$$

where $t \in [0, \Delta T)$ is the time elapsed since the last cache update ($G_c = G_s$), ΔT is the interval to the next cache update; the frequency of cache updates determine how quickly E_d approaches E_c . The interval length to the next cache update, ΔT , is not known a priori, therefore it is estimated using an exponential average function:

$$\Delta T_{n+1} = w_d \Delta T_n + (1 - w_d)\tau_n, \quad (6)$$

where τ_n is the recorded interval for the n^{th} update, ΔT_n is the estimated interval for the n^{th} update, and $w_d \in [0, 1]$ determines whether more weight is given to the previous estimate or the actual reading when computing the next estimate. For $w_c = 1$, the next estimate is based entirely on previous estimates, $\Delta T_{n+1} = \Delta T_n$; for $w_d = 0$, the next estimate becomes the length of the last recorded interval, $\Delta T_{n+1} = \tau_n$. ΔT_0 is set to 100 ms. In an ideal scenario where no network and communication fluctuations are present, a small value of w_d will quickly converge to a constant update interval. Nevertheless, even on an ideal network, the messages exchanged by client and server are still expected to vary in size since the amount of data transferred is proportional to the number of irradiance samples contained in the view frustum of the observer (see §4.1). Empirically, values of w_d in the interval $[0.55, 0.8]$ were found to give the best results and produce less variance in the estimations, in the general case (see §5.1).

4.3 Amortisation of Computation

The virtual scene representation used for rendering is available on both the server and the connected clients. The server needs this information to be able to compute indirect lighting, while the clients require the scene for local rendering, including the reconstruction of indirect lighting and possibly any additional application logic that manipulates the objects within. A client that changes the scene representation by moving objects or lights, for instance, is responsible for informing the server (see Figure 1) and initiating the synchronisation process to ensure the scene is consistent at both ends. The point set Q , which is the representation of diffuse indirect lighting in the virtual scene, is updated once for all connected clients that share the same multi-user environment. Notwithstanding any possible work replication due to scene synchronisation mentioned above, the centralised indirect lighting computation outweighs the penalties thereof; not only are these costs quickly amortised, but the more clients participating in the same virtual environment, the greater the benefits in terms of computation sharing, which result in a form of speed-up.

5 Results

The following results address the scalability of the system, both at client and server-side, bandwidth requirements, latency, and the respective error incurred due to these networking constraints. The system has been tested on four scenes, Sponza Atrium (both the original version and Crytek's), Tony's Barbershop (a Half-Life 2 death match community map), and Conference Room. The Barbershop data set was introduced because besides being a traditional interactive raster scene, it is densely populated with geometric detail from small objects. The generated point cloud representation sizes for the maps were 14.5 k, 21.5 k, 32 k and 38.6 k points respectively. The parameters for the production of these point sets were empirically determined to strike a balance between quality and performance, in terms of both computation and communication. In Figure 7, the cell distribution per sample point in a 32^3 grid is shown; it can be observed that in all four scenes, very few cells contain more than 10 samples. The hardware platform employed as a server for these experiments is equipped with two Intel Xeon E5-2697 CPUs (12 cores per processor), 64 GB of RAM and an NVIDIA GeForce GTX Titan. The client setups range from an ASUS Transformer T100 tablet, equipped with an Intel Atom Z-3740 processor (clocked at 1.33 GHz) and 2 GB of RAM, to an Intel Xeon E5-2643 (4 cores per processor), 16 GB

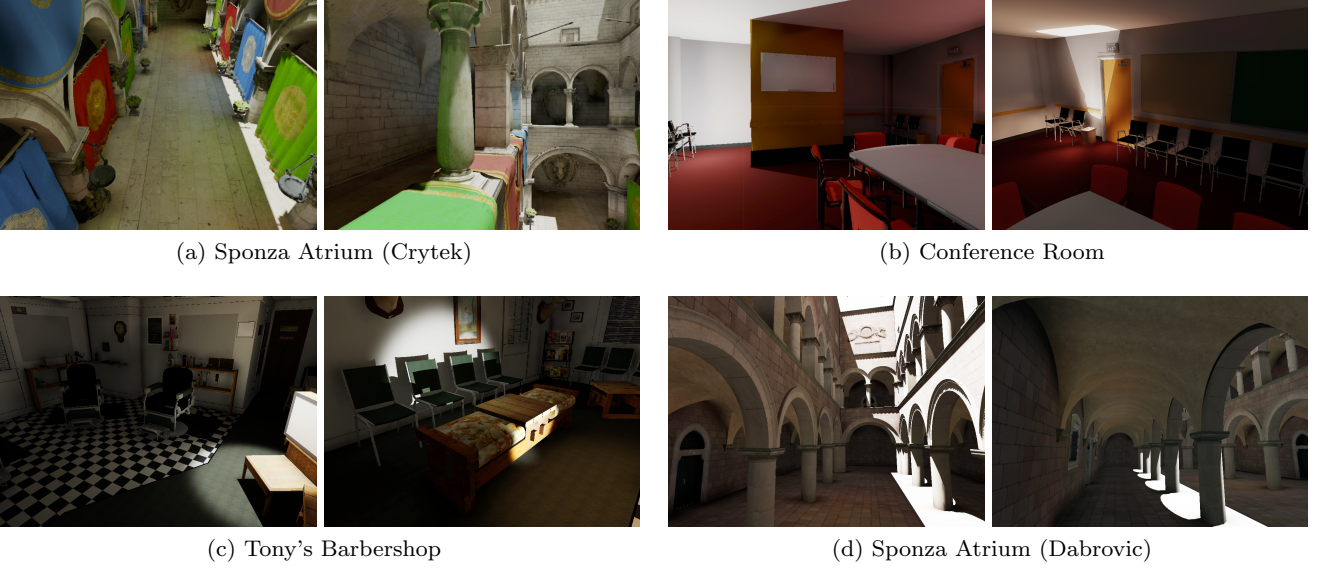


Fig. 6: The scenes used for remote rendering using asynchronous computation.

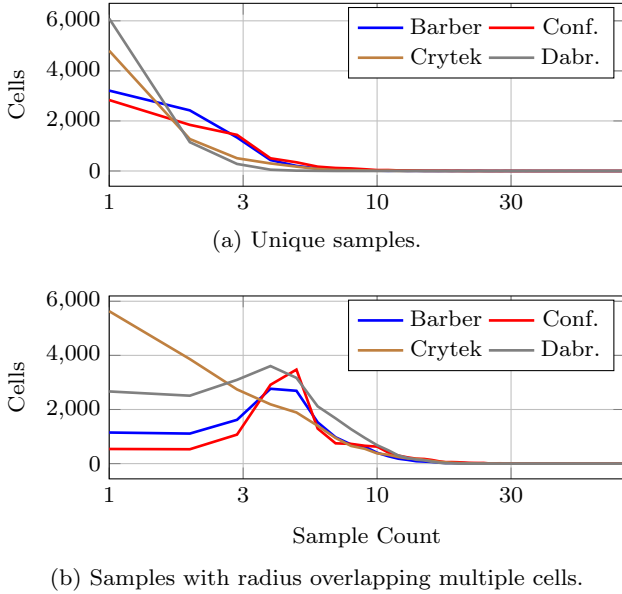


Fig. 7: Plot of cell frequency counts against the number of contained samples in a grid with 32^3 cells. In 7(a), no duplicate (multiple-referenced) samples are shown; in 7(b), samples that overlap multiple cells are included in each of the cells.

of RAM and an NVIDIA GeForce GTX 680 display adapter. The server-side implementation uses NVIDIA OptiX to accelerate VPL shooting and occlusion testing. The client-side implementation has been carried out in Unity3D; reconstruction shaders were written in Direct Compute and thus require Direct3D11-capable hardware.

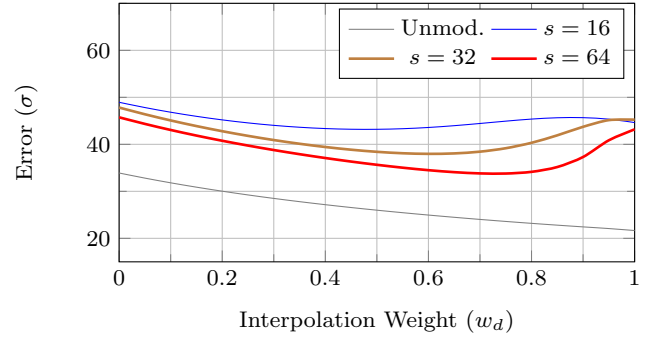


Fig. 8: Standard deviation for prediction error.

5.1 Preliminaries

The values for two-stage smoothing (see §4.2) were set to $w_c = 0.5$ for the first and $w_d = 0.65$ for the second stage respectively. The value for the second stage was determined experimentally; a sequence of one hundred cache update intervals $\tau_i \dots \tau_{i+99}$ was sampled and the estimation error $(\Delta T_n - \tau_n)$ was recorded for $w_d \in \{0, 0.05, 0.1, \dots, 0.95, 1\}$. To simulate fluctuations both in the network and communication, the recorded intervals were modulated by a sinusoidal function:

$$\tau'_n = \tau_n \left(\sin \frac{2n\pi}{s} X_n + 1 \right), \quad (7)$$

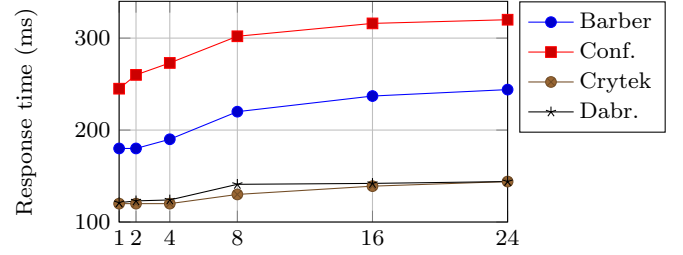
where $s \in \{16, 32, 64\}$ determines the period of the sinusoid, X is a uniformly distributed random sequence with elements in the range $[0, 1)$, and τ_n is the n^{th} recorded interval. Figure 8 shows the variance in the

error for the interval sequences. From the graph, it can be seen that the interpolation weights yielding less variance on average lie in the range $[0.55, 0.8]$. Taking the mean of the error values for each individual weight in the range returns the lowest value at $w_d = 0.65$; this value is used in the following results. The bandwidth (§5.2) and image-fidelity (§5.4) results have been recorded over a scripted set of paths, one for each of the four scenes, since it was necessary to replicate the camera movement over multiple runs of the experiment.

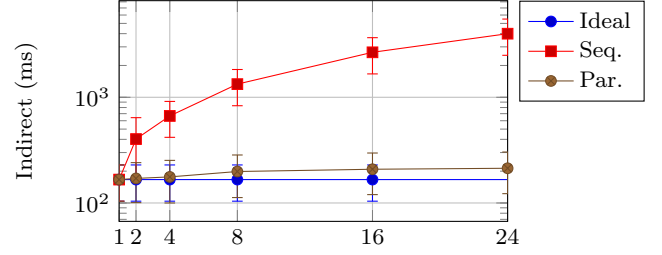
5.2 Bandwidth

The bandwidth requirements of the system have been recorded for all scenes: 2.826 Mbit/s for Barbershop; 1.712 Mbit/s for Conference; 1.54 Mbit/s and 1.159 Mbit/s for Sponza Crytek and Dabrovic respectively. In this test, both the camera view and the light sources followed a scripted path wherein they were constantly changing, precluding any quiescence that could have been achieved by scenes that change very infrequently or in bursts. The client and server complete a roundtrip exchange of indirect lighting at an average frequency of 6 Hz. The total bandwidth requirements do not exceed 3 Mbit/s, which is on the same level as the minimum recommendations for game streaming services (see Table 1), but for the fact that our system is resolution invariant, and reconstructing UHD quality images requires no additional bandwidth.

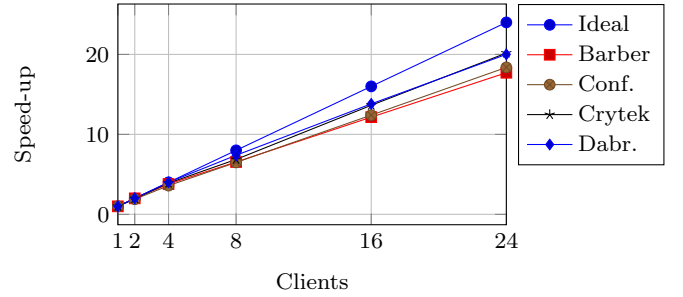
High-dynamic-range (HDR) streaming requires higher bandwidth allocations, depending on the numerical precision allotted to each colour channel; bandwidth requirements double and quadruple for 16-bit (half-float) and 32-bit (single precision) colour ranges respectively. Comparatively, Netflix for instance, requires 25 Mbit/s to stream in HDR, which is twice the estimated bandwidth required by RAIL. It is worth pointing out that although the Barbershop scene has a smaller point cloud than the Conference scene, it is nonetheless more densely populated. This results in a larger number of points captured by frustum culling when compared to the other scenes, which also reflects in the higher bandwidth requirements. Notwithstanding, these requirements are lower than most game streaming requirements for even 720p resolutions. The results highlight the potential of achieving high-fidelity graphics on resources of varying computational power without compromising interactivity response times due to network fluctuations and bandwidth constraints. The method scales adequately at both ends of the hardware spectrum, on average achieving frame rates of approximately 25 Hz at a resolution of 1024×768 on the Intel Atom tablet, and over 60 Hz at UHD



(a) Changes in response times for tested scenes as the number of clients increases - a measure of scalability.



(b) Mean indirect lighting computation scalability for ideal parallel, sequential and actual parallel runs.



(c) Speed-up in the computation of indirect lighting due to amortisation.

Fig. 9: Scalability and performance evaluation results showing system behaviour in terms of response times and indirect lighting computational overhead as the number of connected clients increases.

resolutions on a desktop PC equipped with a GeForce GTX680. The obvious bottleneck during rendering is the reconstruction of indirect lighting, where for each pixel shaded, a nearest neighbour search has to be carried out. Since the system employs a multi-reference grid, the entire set of samples contributing to indirect lighting are contained within a single cell; from Figure 7(b) it becomes evident that the majority of cells contain at most 10 samples.

5.3 Client Scalability (Remote System Overhead)

System scalability is measured in the ability of the system to support multiple connected clients without service degradation. In particular, degradation manifests

in an increased communication latency between clients and the server-end, when streaming indirect lighting. Updates follow a request-response model, where the client initiates each update itself; this request-response cycle has been measured for all test scenes with a varying number of clients and it was found that the increase in latency for up to 24 clients is almost negligible, suggesting that the system is capable of scaling well beyond this number (see Figure 9a). Figure 9b shows the amortisation of computation for the indirect diffuse component, in each of the four scenes. If the computation were to be decentralised and moved back to each individual client, for a homogeneous group of clients c where each member is working individually, the amount of work done would increase by a factor of $c - 1$. For t ms of original computation time on the server, the total work for c independent clients would increase to ct ms. In RAIL, indirect lighting computation is valid for all connected clients, and thus, the ideal computation time would be t as opposed to ct . However, Figure 9a highlights the fact that realisation penalties exist that are introduced due to communication and synchronisation, and increase as more clients are added. To account for them, the actual computation time for c clients becomes $r_p(c) + t$, where $r_p(c)$ is the realisation penalty for c clients. Figure 9b shows mean plots for t (Ideal), $r_p(c) + t$ (Par.) and ct (Seq.) for all four scenes. Figure 9c expresses this gain in terms of computation speed-up; it must be stressed that the gain comes at no cost since the clients would otherwise still have to perform the computation of indirect lighting themselves.

5.4 Image fidelity

The remote asynchronous nature of RAIL introduces temporal discrepancies between the direct and indirect lighting components. In this test we measure image fidelity as a function of these discrepancies; particularly, we measure the difference between a typical and a zero-latency execution of the system, the latter generated entirely and synchronously on the server using the rendering method described in Section 3, without any time constraints. For both methods, scripted walkthroughs over three of the test scenes were rendered and compared. The light sources and camera view change throughout all but the end of the animation, where the image was allowed to converge over a number of frames. Figure 10 shows two selected image pairs from the generated walkthroughs; their structured similarity (SSIM) index is 0.9429 and 0.8165 for the left and right pair respectively. The peak signal-to-noise ratio (PSNR) was computed for each pair of frames in the resulting animations; this is shown in Figure 11.

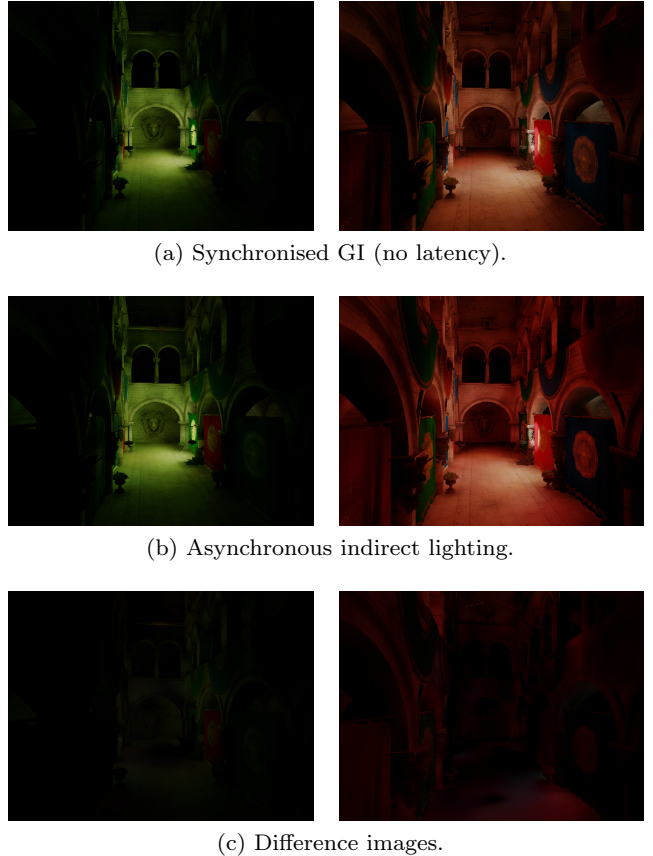


Fig. 10: A comparison between zero-latency (10a) and asynchronous indirect lighting (10b). The latter contributes to output lag, where the indirect lighting appears to trail behind the direct lighting. The differences between the two pair of images are shown in Figure 10c. The structural similarity (SSIM) index is 0.9429 for the left image pair and 0.8165 for the right.

5.5 Limitations and future directions

In the constructed prototype, indirect lighting is computed for diffuse surfaces and stored in terms of irradiance; a logical avenue to be pursued by future work is the extension of the system to include support for glossy materials. Dynamic objects, although supported, only provide a coarse approximation to indirect lighting through the ambient grid. Besides investigating other methods for dynamic objects, a unified solution for static and dynamic geometry could be sought. The system currently operates under the assumption that a single point set and the respective grid-based acceleration structure are sufficient to model a scene; however, large complex scenes with sparse geometry layouts may benefit from some overlay structure, such as a graph, that links together multiple grids for a more efficient representation. As a note on

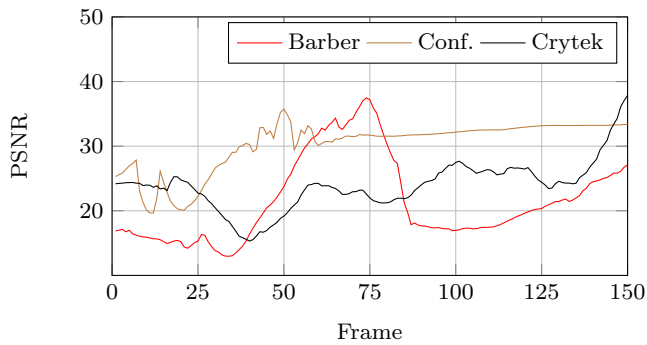


Fig. 11: PSNR values characterising latency of indirect lighting over animation sequences for tested scenes.

the generated sizes of point sets, an overly dense data set may result in higher bandwidth usage as well as longer computation times at the server-end, for the estimation of indirect lighting. This is suggested by the bandwidth usage and the request-response cycle times of Figure 9a. The grid implementation itself is straightforward; the memory arrangement avoids interleaving of data to speed up copies during updates. However, during the reconstruction phase, when cells are constantly being queried, memory access patterns have not been profiled to detect any sub-optimal behaviour. The process could be further studied to see if there is still potential for optimisation. Since scene point-cloud generation is a one-time precomputation step, dart throwing and naïve Poisson disk sampling were employed [6]; however, other more efficient strategies and approaches could be used, with desirable properties such as the generation of maximal distributions [18]. Blue noise sampling strategies [40], [3], [37] may also be investigated due to their improved convergence rate over other samplers, including Poisson disk sampling, during Monte Carlo integration [36]; the application of these strategies to the domains of geometry representation and reconstruction may lead towards improved solutions that yield better coverage or faster convergence.

6 Conclusion

The results highlight the potential of achieving high-fidelity graphics on resources of varying computational power without compromising interactivity response times due to network fluctuations and bandwidth constraints. Server-side scalability is very promising, with minimal overhead incurred when increasing the number of clients. The scalability results in Figure 9b show that additional clients in multi-user environments can be added at very little cost, since indirect lighting

is amortised over them. This carries a significant advantage over streaming solutions which provide each of the clients with rendering sandboxes that do not interact and thus, share no computation load. While other cloud methods have been presented, both as fully streaming solutions and as distributed rendering pipelines, our solution requires lower bandwidth and is robust to latency. Furthermore, with respect to purely streaming solutions, our method can amortise the computation of indirect lighting in multi-user environments with minimal costs for each additional client.

References

- Autodesk 360 (2014). URL <http://www.autodesk.com/products/rendering/overview>
- renderrocket (2014). URL <http://www.renderrocket.com/features/>
- Ahmed, A.G., Niese, T., Huang, H., Deussen, O.: An adaptive point sampler on a regular lattice. *ACM Transactions on Graphics (TOG)* **36**(4), 138 (2017)
- Bashford-Rogers, T., Debattista, K., Chalmers, A.: Importance driven environment map sampling (2013)
- Bierton, D.: Face-off: Gaikai vs. onlive (2012). URL <http://www.eurogamer.net/articles/digitalfoundry-face-off-gaikai-vs-onlive>
- Bikker, J., Reijerse, R.: A precalculated point set for caching shading information. In: *Eurographics 2009-Short Papers*, pp. 65–68. The Eurographics Association (2009)
- Brouillat, J., Gautron, P., Bouatouch, K.: Photon-driven irradiance cache. In: *Computer Graphics Forum*, vol. 27, pp. 1971–1978. Wiley Online Library (2008)
- Bugeja, K., Debattista, K., Spina, S., Chalmers, A.: Collaborative high-fidelity rendering over peer-to-peer networks. In: *Eurographics Symposium on Parallel Graphics and Visualization*, pp. 9–16. The Eurographics Association (2014)
- Chalmers, A., Reinhard, E., Davis, T.: *Practical parallel rendering*. CRC Press (2002)
- Crassin, C., Luebke, D., Mara, M., McGuire, M., Oster, B., Shirley, P., Sloan, P.P., Wyman, C.: Cloud-Light: A system for amortizing indirect lighting in real-time rendering. *Journal of Computer Graphics Techniques (JCGT)* **4**(4), 1–27 (2015). URL <http://jcgt.org/published/0004/04/01/>
- Crassin, C., Neyret, F., Sainz, M., Green, S., Eisemann, E.: Interactive indirect illumination using voxel cone tracing. In: *Computer Graphics Forum*, vol. 30, pp. 1921–1930. Wiley Online Library (2011)
- Dachsbacher, C., Krivánek, J., Hašan, M., Arbree, A., Walter, B., Novák, J.: Scalable realistic rendering with many-light methods. In: *Computer Graphics Forum*, vol. 33, pp. 88–104. Wiley Online Library (2014)
- Dachsbacher, C., Stamminger, M.: Reflective shadow maps. In: *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pp. 203–231. ACM (2005)
- Dammert, H., Keller, A., Lensch, H.P.: Progressive point-light-based global illumination. In: *Computer Graphics Forum*, vol. 29, pp. 2504–2515. Wiley Online Library (2010)

15. Debattista, K., Dubla, P., Banterle, F., Santos, L.P., Chalmers, A.: Instant caching for interactive global illumination. In: *Computer Graphics Forum*, vol. 28, pp. 2216–2228. Wiley Online Library (2009)
16. Deering, M., Winner, S., Schediwy, B., Duffy, C., Hunt, N.: The triangle processor and normal vector shader: a vlsi system for high performance graphics. In: *ACM SIGGRAPH Computer Graphics*, vol. 22, pp. 21–30. ACM (1988)
17. Dippé, M.A., Wold, E.H.: Antialiasing through stochastic sampling. *ACM Siggraph Computer Graphics* **19**(3), 69–78 (1985)
18. Gamito, M.N., Maddock, S.C.: Accurate multidimensional poisson-disk sampling. *ACM Transactions on Graphics (TOG)* **29**(1), 8 (2009)
19. Jensen, H.W.: *Realistic image synthesis using photon mapping*. AK Peters, Ltd. (2001)
20. Kajiya, J.T.: The rendering equation. In: *ACM Siggraph Computer Graphics*, vol. 20, pp. 143–150. ACM (1986)
21. Kaplanyan, A., Dachsbacher, C.: Cascaded light propagation volumes for real-time indirect illumination. In: *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pp. 99–107. ACM (2010)
22. Keller, A.: Instant radiosity. In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 49–56. ACM Press/Addison-Wesley Publishing Co. (1997)
23. Kopf, J., Cohen, M.F., Lischinski, D., Uyttendaele, M.: Joint bilateral upsampling. In: *ACM Transactions on Graphics (TOG)*, vol. 26, p. 96. ACM (2007)
24. Langer, M.S., Bülthoff, H.H.: Depth discrimination from shading under diffuse lighting. *Perception* **29**(6), 649–660 (1999)
25. Lehmann, M.A.: Lzf compression library (liblzf) (2014). URL <http://www.goof.com/pcg/marc/liblzf.html>
26. Lewis, M.: The new cards. *Communications of the ACM* **45**(1), 30–31 (2002)
27. Liu, C., Jia, J., Zhang, Q., Zhao, L.: Lightweight web-sim rendering framework based on cloud-baking. In: *Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pp. 221–229. ACM (2017)
28. Manzano, M., Hernández, J.A., Uruenña, M., Calle, E.: An empirical study of cloud gaming. In: *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*, p. 17. IEEE Press (2012)
29. Mara, M., Luebke, D., McGuire, M.: Toward practical real-time photon mapping: efficient gpu density estimation. In: *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pp. 71–78. ACM (2013)
30. McGuire, M., Luebke, D.: Hardware-accelerated global illumination by image space photon mapping. In: *Proceedings of the 2009 ACM SIGGRAPH/EuroGraphics conference on High Performance Graphics*. ACM, New York, NY, USA (2009). URL <http://graphics.cs.williams.edu/papers/PhotonHPG09/>
31. Mitchell, J., McTaggart, G., Green, C.: Shading in valve's source engine. In: *ACM SIGGRAPH 2006 Courses*, pp. 129–142. ACM (2006)
32. Mittring, M.: Finding next gen: Cryengine 2. In: *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, pp. 97–121. ACM, New York, NY, USA (2007). DOI 10.1145/1281500.1281671. URL <http://doi.acm.org/10.1145/1281500.1281671>
33. Myszkowski, K., Tawara, T., Akamine, H., Seidel, H.P.: Perception-guided global illumination solution for animation rendering. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 221–230. ACM (2001)
34. Pajak, D., Herzog, R., Eisemann, E., Myszkowski, K., Seidel, H.P.: Scalable remote rendering with depth and motion-flow augmented streaming. In: *Computer Graphics Forum*, vol. 30, pp. 415–424. Wiley Online Library (2011)
35. Pál, L., Oláh-Gál, R., Makó, Z.: Shepard interpolation with stationary points. *Acta Univ. Sapientiae* **1**(1), 5–13 (2009)
36. Pilleboue, A., Singh, G., Coeurjolly, D., Kazhdan, M., Ostromoukhov, V.: Variance analysis for monte carlo integration. *ACM Transactions on Graphics (TOG)* **34**(4), 124 (2015)
37. Qin, H., Chen, Y., He, J., Chen, B.: Wasserstein blue noise sampling. *ACM Transactions on Graphics (TOG)* **36**(5), 168 (2017)
38. Saito, T., Takahashi, T.: Comprehensible rendering of 3-d shapes. In: *ACM SIGGRAPH Computer Graphics*, vol. 24, pp. 197–206. ACM (1990)
39. Shepard, D.: A two-dimensional interpolation function for irregularly-spaced data. In: *Proceedings of the 1968 23rd ACM national conference*, pp. 517–524. ACM (1968)
40. Wachtel, F., Pilleboue, A., Coeurjolly, D., Breeden, K., Singh, G., Cathelin, G., De Goes, F., Desbrun, M., Ostromoukhov, V.: Fast tile-based adaptive sampling with user-specified fourier spectra. *ACM Transactions on Graphics (TOG)* **33**(4), 56 (2014)
41. Walter, B., Drettakis, G., Parker, S.: Interactive rendering using the render cache. In: *Rendering techniques 99*, pp. 19–30. Springer (1999)
42. Ward, G.J., Rubinstein, F.M., Clear, R.D.: A ray tracing solution for diffuse interreflection. *ACM SIGGRAPH Computer Graphics* **22**(4), 85–92 (1988)